3차시 과제 풀이

문제 목록

문제 번호	문제 이름	출처
BOJ 1822	차집합	
BOJ 20920	영단어 암기는 괴로워	
BOJ 1991	트리 순회	
BOJ 7662	이중 우선순위 큐	2013 ICPC 인터넷 예선
BOJ 1976	여행 가자	
BOJ 1351	무한 수열	
BOJ 15942	Thinking Heap	UCPC 2018
BOJ 2696	중앙값 구하기	2009 뉴욕 리저널
BOJ 17435	합성함수와 쿼리	
BOJ 11438	LCA 2	
BOJ 2957	이진 탐색 트리	COCI 2008/2009 #3
BOJ 13306	트리	KOI 2016 중등부
BOJ 17469	트리의 색깔과 쿼리	**내가 만든 문제**
BOJ 13303	장애물 경기	KOI 2016 초등부
BOJ 13511	트리와 쿼리 2	
BOJ 21607	Polynomial and Easy Queries	
BOJ 20030	트리와 쿼리 17	2020 KAIST Mock

BOJ 1822 차집합

set에서 어떤 원소가 존재하는지 판별하는 것은 `set::find`의 반환값이 `std::end`가 아닌지 확인하면 된다.
`set::count`의 반환값이 0이 아닌지 확인하는 방법도 있지만, multiset에서는 사용하면 시간 초과가 발생할 수 있으니 조심하자.

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N, M; cin >> N >> M;
    set<int> A;
    for(int i=0; i<N; i++){
```

```
        int x; cin >> x;
        A.insert(x);
    }
    for(int i=0; i<M; i++){
        int x; cin >> x;
        if(A.find(x) != A.end()) A.erase(x);
        // if(A.count(x)) A.erase(x);
    }
    cout << A.size() << "\n";
    for(auto i : A) cout << i << " ";
}
```

BOJ 20920 영단어 암기는 괴로워

문자열의 등장 횟수는 `map<string, int>`를 이용하면 편하제 저장할 수 있다. 비교 함수를 잘 작성해서 정렬하자.

```
#include <bits/stdc++.h>
using namespace std;

int N, M;
vector<string> V;
map<string, int> C;

bool Compare(const string &a, const string &b){
    if(C[a] != C[b]) return C[a] > C[b];
    if(a.size() != b.size()) return a.size() > b.size();
    return a < b;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=0; i<N; i++){
        string s; cin >> s;
        if(s.size() >= M) V.push_back(s), C[s]++;
    }
    sort(V.begin(), V.end(), Compare);
    for(int i=0; i<V.size(); i++){
        if(i == 0 || V[i-1] != V[i]) cout << V[i] << "\n";
    }
}
```

BOJ 1991 트리 순회

열심히 구현하자.

```
#include <bits/stdc++.h>
using namespace std;

int N, G[26][2];

void pre_order(int v){
    cout << char(v + 'A');
    if(G[v][0]) pre_order(G[v][0]);
    if(G[v][1]) pre_order(G[v][1]);
```

```
}
void in_order(int v){
    if(G[v][0]) in_order(G[v][0]);
    cout << char(v + 'A');
    if(G[v][1]) in_order(G[v][1]);
}
void post_order(int v){
    if(G[v][0]) post_order(G[v][0]);
    if(G[v][1]) post_order(G[v][1]);
    cout << char(v + 'A');
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=0; i<N; i++){
        char p, l, r; cin >> p >> l >> r;
        if(l != '.') G[p-'A'][0] = l-'A';
        if(r != '.') G[p-'A'][1] = r-'A';
    }
    pre_order(0); cout << "\n";
    in_order(0); cout << "\n";
    post_order(0); cout << "\n";
}
```

BOJ 7662 이중 우선순위 큐

multiset을 사용하자.

multiset에서 어떤 값을 지울 때 `erase(value)`를 호출하면 `value`와 일치하는 **모든 원소**가 사라진다는 것을 주의하자. `find` 등으로 iterator를 찾아서 호출해야 한다.

```
#include <bits/stdc++.h>
using namespace std;

void Solve(){
    int Q; cin >> Q;
    multiset<int> st;
    while(Q--){
        char op; int v; cin >> op >> v;
        if(op == 'I') st.insert(v);
        else if(op == 'D' && v == -1 && !st.empty()) st.erase(st.begin());
        else if(op == 'D' && v == +1 && !st.empty()) st.erase(prev(st.end()));
    }
    if(st.empty()) cout << "EMPTY\n";
    else cout << *st.rbegin() << " " << *st.begin() << "\n";
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int T; cin >> T;
    while(T--) Solve();
}
```

BOJ 1976 여행 가자

서로 이동 가능한 정점들을 Union-Find를 이용해 묶어주면 된다.

```
#include <bits/stdc++.h>
using namespace std;

int P[222];
int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
void Union(int u, int v){ if(Find(u) != Find(v)) P[Find(u)] = Find(v); }

int N, M, A[1010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    iota(P+1, P+N+1, 1); // for(int i=1; i<=N; i++) P[i] = i;
    for(int i=1; i<=N; i++){
        for(int j=1; j<=N; j++){
            int v; cin >> v;
            if(v == 1) Union(i, j);
        }
    }
    for(int i=1; i<=M; i++) cin >> A[i];

    bool flag = true;
    for(int i=2; i<=M; i++) if(Find(A[i-1]) != Find(A[i])) flag = false;
    if(flag) cout << "YES";
    else cout << "NO";
}
```

BOJ 1351 무한 수열

단순한 DP 문제인데 N이 10^{12}까지 커질 수 있다는 것이 문제다.

잘 생각해보면 $f(N)$을 구할 때 실제로 사용하게 되는 $f(x)$의 값들이 많지 않다는 것을 알 수 있고, 그러므로 Top-Down 방식으로 필요한 수들만 보면 된다. DP Table을 map으로 구현하면 된다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

ll N, P, Q;
map<ll, ll> D;

ll f(ll x){
    if(x == 0) return 1;
    if(D.find(x) != D.end()) return D[x];
    return D[x] = f(x/P) + f(x/Q);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> P >> Q;
    cout << f(N);
}
```

BOJ 15942 Thinking Heap

미리 min-heap을 만든 다음 차례대로(BFS 혹은 인덱스 순서대로) 넣어주면 그대로 min-heap이 만들어진다는 것을 알 수 있다. 그러므로 P번째 인덱스에 K가 오도록 하는 min-heap을 만들면 된다.

P의 조상들의 값은 항상 K보다 작기 때문에, 루트 정점부터 P의 부모 정점까지의 경로를 따라 차례대로 $1, 2, 3, \cdots$를 배정하자.

P의 자손들의 값은 항상 K보다 크기 때문에, DFS 등을 이용해 P의 자손들에 $N, N-1, N-2, \cdots$를 배정하자.

나머지 정점들은 남은 수들을 오름차순으로 배정해주면 된다.

```
#include <bits/stdc++.h>
using namespace std;

int N, K, P, Heap[202020];
int Small, Large;

void FillSmall(){
    vector<int> up;
    for(int i=P/2; i; i/=2) up.push_back(i);
    reverse(up.begin(), up.end());
    for(auto i : up) Heap[i] = Small++;
}

void FillLarge(int v){
    if(v*2 <= N) FillLarge(v*2);
    if(v*2+1 <= N) FillLarge(v*2+1);
    Heap[v] = Large--;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K >> P;
    Small = 1; Large = N;
    Heap[P] = K;

    if(P != 1) FillSmall();
    if(P*2 <= N) FillLarge(P*2);
    if(P*2+1 <= N) FillLarge(P*2+1);

    if(Small > K || Large < K){
        cout << -1;
        return 0;
    }

    for(int i=1; i<=N; i++){
        if(Heap[i]) continue;
        if(Small == K) Small++;
        Heap[i] = Small++;
    }

    for(int i=1; i<=N; i++) cout << Heap[i] << "\n";
}
```

BOJ 2696 중앙값 구하기

값이 서로 다른 원소가 홀수 개 있을 때 어떤 값 m이 중앙값이라는 것은, (m보다 작은 원소의 개수)와 (m보다 큰 원소의 개수)가 동일하다는 것을 의미한다. 값이 동일한 원소가 있더라도 m 이하의 원소, m 이상의 원소로 비슷하게 생각할 수 있다.

여기서 알 수 있는 사실은, 현재까지의 원소 개수 N과 중앙값 m에 대해 (m보다 작거나 같은 원소 $\frac{N-1}{2}$개), (m보다 크거나 같은 원소 $\frac{N-1}{2}$개)를 적당한 자료구조를 이용해 관리하면 원소가 추가되더라도 중앙값을 쉽게 구할 수 있다는 것이다.

전자를 max-heap, 후자를 min-heap으로 관리하면 $O(N \log N)$에 문제를 해결할 수 있다.

```
#include <bits/stdc++.h>
using namespace std;

void Solve(){
    int median;
    priority_queue<int, vector<int>, less<int>> max_heap;
    priority_queue<int, vector<int>, greater<int>> min_heap;

    int N, cnt = 0; cin >> N;
    cout << (N+1) / 2 << "\n";

    cin >> median;
    cout << median << " "; cnt++;
    for(int i=2; i<=N; i+=2){
        int a, b; cin >> a >> b;
        if(a < median) max_heap.push(a);
        else min_heap.push(a);
        if(b < median) max_heap.push(b);
        else min_heap.push(b);

        while(max_heap.size() != min_heap.size()){
            if(max_heap.size() >  min_heap.size()){
                min_heap.push(median);
                median = max_heap.top(); max_heap.pop();
            }
            else{
                max_heap.push(median);
                median = min_heap.top(); min_heap.pop();
            }
        }
        cout << median << " "; cnt++;
        if(cnt % 10 == 0) cout << "\n";
    }
    if(cnt % 10 != 0) cout << "\n";
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int T; cin >> T;
    while(T--) Solve();
}
```

BOJ 17435 합성함수와 쿼리

Sparse Table 연습 문제

```
#include <bits/stdc++.h>
using namespace std;

int N, Q, T[20][202020];

int Get(int n, int x){
    for(int i=0; n; i++, n>>=1) if(n & 1) x = T[i][x];
    return x;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> T[0][i];
    for(int i=1; i<20; i++) for(int j=1; j<=N; j++) T[i][j] = T[i-1][T[i-1][j]];
    cin >> Q;
    while(Q--){
        int n, x; cin >> n >> x;
        cout << Get(n, x) << "\n";
    }
}
```

BOJ 11438 LCA 2

LCA 연습 문제

```
#include <bits/stdc++.h>
using namespace std;

int N, Q, D[101010], P[22][101010];
vector<int> G[101010];

void DFS(int v, int b=-1){
    for(auto i : G[v]){
        if(i == b) continue;
        D[i] = D[v] + 1; P[0][i] = v;
        DFS(i, v);
    }
}

int LCA(int u, int v){
    if(D[u] < D[v]) swap(u, v);
    int diff = D[u] - D[v];
    for(int i=0; diff; i++, diff>>=1) if(diff & 1) u = P[i][u];
    if(u == v) return u;
    for(int i=21; i>=0; i--) if(P[i][u] != P[i][v]) u = P[i][u], v = P[i][v];
    return P[0][u];
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
```

```
    for(int i=1; i<N; i++){
        int s, e; cin >> s >> e;
        G[s].push_back(e); G[e].push_back(s);
    }
    DFS(1);
    for(int i=1; i<22; i++) for(int j=1; j<=N; j++) P[i][j] = P[i-1][P[i-1][j]];
    cin >> Q;
    while(Q--){
        int u, v; cin >> u >> v;
        cout << LCA(u, v) << "\n";
    }
}
```

BOJ 2957 이진 탐색 트리

BST에 원소를 주어진 순서대로 삽입할 때, 삽입한 원소들의 깊이의 합을 구하는 문제다.

어떤 수 x가 BST에 들어가는 상황을 생각해보자. 그림을 그려가며 잘 생각해보면 두 가지 경우 중 하나에 해당한다는 것을 알 수 있다.

- x보다 큰 수 중에서 최솟값의 자식으로 들어감
- x보다 작은 수 중에서 최댓값의 자식으로 들어감

x보다 큰 수 중 가장 작은 값과 x보다 작은 수 중 가장 큰 값을 각각 u, v라고 하면, x의 깊이 D_x는 $D_x = \max(D_u, D_v) + 1$로 계산할 수 있다.

lower_bound와 upper_bound를 적절히 잘 이용하자.

```
#include <bits/stdc++.h>
using namespace std;

int N, D[303030];
long long C;
set<int> S;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++){
        int t; cin >> t;
        if(i == 1){
            S.insert(t); D[t] = 0;
            cout << C << "\n";
            continue;
        }
        auto it = S.upper_bound(t);
        if(it != S.end()) D[t] = max(D[t], D[*it] + 1);
        if(it != S.begin()) D[t] = max(D[t], D[*prev(it)] + 1);
        S.insert(t); C += D[t];
        cout << C << "\n";
    }
}
```

BOJ 13306 트리

간선을 끊는 쿼리는 반대로 생각하면 간선을 연결하는 쿼리라고 생각할 수 있다. 쿼리의 순서를 뒤집으면 Union-Find 문제로 바뀌게 된다.

이렇게 쿼리를 모두 입력받은 다음, 쿼리 순서를 자유롭게 바꿔서 처리하는 방법을 오프라인 방식이라고 부른다.

https://solved.ac/problems/tags/offline_queries

```
#include <bits/stdc++.h>
using namespace std;

int P[202020];
int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
void Merge(int u, int v){ if(Find(u) != Find(v)) P[Find(u)] = Find(v); }

int N, Q, G[202020];
int A[404040], B[404040], C[404040];
vector<int> R;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> Q; Q += N-1;
    for(int i=2; i<=N; i++) cin >> G[i];
    for(int i=1; i<=Q; i++){
        cin >> A[i];
        if(A[i] == 0) cin >> B[i];
        else cin >> B[i] >> C[i];
    }

    iota(P+1, P+N+1, 1);
    for(int i=Q; i>=1; i--){
        if(A[i] == 0) Merge(B[i], G[B[i]]);
        else R.push_back(Find(B[i]) == Find(C[i]));
    }

    reverse(R.begin(), R.end());
    for(auto i : R) cout << (i ? "YES" : "NO") << "\n";
}
```

BOJ 17469 트리의 색깔과 쿼리

BOJ 13306 트리 문제에서 "정점의 색깔"이라는 개념만 추가된 것이다. 색깔의 종류는 set을 이용하면 쉽게 관리할 수 있다. 즉, Union-Find의 Union에서 두 컴포넌트의 색깔을 관리하고 있는 set을 합쳐야 한다.

set을 합칠 때 작은 set에 있는 원소를 큰 set으로 옮겨주면(Small to Large) 원소들이 최대 $O(N \log N)$번만 이동해서 $O(N \log^2 N)$에 문제를 풀 수 있다.

```
#include <bits/stdc++.h>
using namespace std;

int P[101010];
set<int> S[101010];
int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
```

```
void Merge(int u, int v){
    u = Find(u); v = Find(v);
    if(u == v) return;
    if(S[u].size() > S[v].size()) swap(u, v);
    for(auto i : S[u]) S[v].insert(i); S[u].clear();
    P[u] = v;
}

int N, Q, G[101010], A[1110101], B[1110101];
vector<int> R;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> Q; Q += N-1;
    for(int i=2; i<=N; i++) cin >> G[i];
    for(int i=1,t; i<=N; i++) cin >> t, S[i].insert(t);

    iota(P+1, P+N+1, 1);
    for(int i=1; i<=Q; i++) cin >> A[i] >> B[i];
    for(int i=Q; i>=1; i--){
        if(A[i] == 1) Merge(B[i], G[B[i]]);
        else R.push_back(S[Find(B[i])].size());
    }
    reverse(R.begin(), R.end());
    for(auto i : R) cout << i << "\n";
}
```

BOJ 13303 장애물 경기

장애물을 만날 때마다 장애물의 양쪽 끝점에 대해 (끝점의 좌표, 이동 거리)를 관리하면 된다. 이때, 가로 방향으로의 전체 이동 거리는 항상 일정하기 때문에 세로 방향으로의 이동 거리만 관리하면 된다.

장애물을 x좌표 기준으로 정렬해서 차례대로 처리하자. 장애물을 만날 때마다 위/아래 2가지로 분리되기 때문에, 단순하게 정보를 관리하면 2^N개의 정보를 관리해야 돼서 문제를 해결할 수 없다.

세로 방향으로의 이동 거리만 관리하면 되기 때문에, 장애물의 양쪽 끝점에 대해 **(y좌표, 이동 거리)** pair만 관리해도 된다는 사실을 알 수 있다. 그러므로 y좌표 별로 세로 방향 이동 거리가 가장 짧은 것만 들고 있으면 최대 $2N$개의 pair만 관리하면 된다.

열린 구간 (y_1, y_2)를 커버하는 장애물을 만났을 경우, 현재 들고 있는 (y좌표, 이동 거리) pair 중 $y_1 < y < y_2$인 모든 pair를 없애주고, 이들이 y_1으로 가는 경우와 y_2로 가는 경우를 추가로 관리하면 된다. 이러한 작업은 set을 이용해 쉽게 할 수 있다.

자세한 구현은 아래 코드를 참고하자.

```
#include <bits/stdc++.H>
using namespace std;
using ll = long long;
constexpr ll INF = 0x3f3f3f3f3f3f3f3f;

struct Hurdle{
    int x, s, e;
    bool operator < (const Hurdle &hurdle) const { return x < hurdle.x; }
};

int N, SY, EX;
Hurdle A[101010];
```

```
ll D[2020202], R = INF;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> SY >> EX;
    for(int i=1; i<=N; i++) cin >> A[i].x >> A[i].s >> A[i].e;
    sort(A+1, A+N+1);

    set<int> S;
    memset(D, 0x3f, sizeof D);
    S.insert(SY); D[SY] = 0;

    for(int i=1; i<=N; i++){
        int s = A[i].s, e = A[i].e;
        for(auto it=S.upper_bound(s); it!=S.end() && *it<e; it=S.erase(it)){
            D[s] = min(D[s], D[*it] + *it - s);
            D[e] = min(D[e], D[*it] + e - *it);
            D[*it] = INF; // erase
        }
        if(D[s] != INF) S.insert(s);
        if(D[e] != INF) S.insert(e);
    }

    for(auto i : S) R = min(R, D[i] + EX);
    cout << R << "\n";

    vector<int> V;
    for(auto i : S) if(D[i]+EX == R) V.push_back(i);
    cout << V.size() << " ";
    for(auto i : V) cout << i << " ";
}
```

BOJ 13511 트리와 쿼리 2

루트 정점에서 u로 가는 경로의 깅리를 C_u라고 하자. u에서 v로 가는 경로의 길이는 $C_u + C_v - 2 \cdot C_{LCA(u,v)}$이다. 그러므로 1번 쿼리는 쉽게 해결할 수 있다.

2번 쿼리는 Sparse Table을 열심히 쓰면 된다. LCA를 넘어가는 경우와 넘어가지 않는 경우로 나눠서 처리하면 편하다.

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using PLL = pair<ll, ll>;

int N, Q, P[22][101010], D[101010];
ll C[101010];
vector<PLL> G[101010];

void DFS(int v, int b=-1){
    for(auto i : G[v]){
        if(i.x == b) continue;
        P[0][i.x] = v;
        D[i.x] = D[v] + 1;
        C[i.x] = C[v] + i.y;
```

```
        DFS(i.x, v);
    }
}

int LCA(int u, int v){
    if(D[u] < D[v]) swap(u, v);
    int diff = D[u] - D[v];
    for(int i=0; diff; i++, diff>>=1) if(diff & 1) u = P[i][u];
    if(u == v) return u;
    for(int i=21; ~i; i--) if(P[i][u] != P[i][v]) u = P[i][u], v = P[i][v];
    return P[0][u];
}
ll Dist(int u, int v){ return C[u] + C[v] - 2 * C[LCA(u, v)]; }

int Kth(int u, int v, int k){
    int l = LCA(u, v); k--;
    if(k > D[u] - D[l]){
        k = D[v] - D[l] - k + D[u] - D[l];
        u = v;
    }
    for(int i=0; k; i++, k>>=1) if(k & 1) u = P[i][u];
    return u;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<N; i++){
        int s, e, x; cin >> s >> e >> x;
        G[s].emplace_back(e, x);
        G[e].emplace_back(s, x);
    }
    DFS(1);
    for(int i=1; i<22; i++) for(int j=1; j<=N; j++) P[i][j] = P[i-1][P[i-1][j]];

    cin >> Q;
    while(Q--){
        int op, a, b, c; cin >> op >> a >> b;
        if(op == 1) cout << Dist(a, b) << "\n";
        else cin >> c, cout << Kth(a, b, c) << "\n";
    }
}
```

BOJ 21607 Polynomial and Easy Queries

놀랍게도 $f(g(x)) = g(f(x))$이다. 그러므로 수열 A의 초기 상태에서 A_i에 f, g를 적용한 횟수만 알고 있으면 Sparse Table을 이용해 답을 구할 수 있다.

A_i에 f, g를 적용한 횟수는 Fenwick Tree나 Segment Tree 등을 이용해 쉽게 관리할 수 있다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr int S = 505050;
constexpr int M = 100003;

struct FenwickTree{
```

```
    ll T[S];
    void _add(int x, int v){ for(x+=2; x<S; x+=x&-x) T[x] += v; }
    void add(int s, int e, int v=1){ _add(s, v); _add(e+1, -v); }
    ll get(int x){ ll ret = 0; for(x+=2; x>0; x-=x&-x) ret += T[x]; return ret;
}
};
struct SparseTable{
    int T[22][M];
    void set(int x, int v){ T[0][x] = v; }
    void build(){
        for(int i=1; i<22; i++) for(int j=0; j<M; j++) T[i][j] = T[i-1][T[i-1]
[j]];
    }
    int get(int x, int v){
        for(int i=0; v; i++, v>>=1) if(v & 1) x = T[i][x];
        return x;
    }
};

inline int f(ll x){ return (2*x*x - 1 + M) % M; }
inline int g(ll x){ return (4*x*x - 3 + M) * x % M; }

int N, Q, A[S];
FenwickTree Tf, Tg;
SparseTable Sf, Sg;

int Get(int x, int a, int b){
    return Sg.get(Sf.get(x, a), b);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> Q;
    for(int i=1; i<=N; i++) cin >> A[i];
    for(int i=0; i<M; i++) Sf.set(i, f(i)), Sg.set(i, g(i));
    Sf.build(); Sg.build();
    while(Q--){
        int op, a, b; cin >> op >> a;
        if(op == 3) cout << Get(A[a], Tf.get(a), Tg.get(a)) << "\n";
        else cin >> b, (op == 1 ? Tf : Tg).add(a, b);
    }
}
```

BOJ 20030 트리와 쿼리 17

정점의 가중치가 바뀌는 상황에서 Weighted Centroid를 구하는 문제다.

1번 쿼리는 Euler Tour Trick + Segment Tree Lazy Propagation, 2번 쿼리는 Heavy Light Decomposition + Segment Tree Lazy Propagation으로 처리할 수 있다.

Euler Tour Trick을 이용해 트리를 일자로 펼치자. Weighted Centroid를 루트로 하는 서브 트리는 항상 Prefix Sum의 median 지점을 포함한다. 그러므로 median 지점에서 시작해서, 조상을 쭉 보면서 Weighted Centroid를 찾으면 된다.

당연히 조상 정점들을 보는 건 Sparse Table을 이용해 $O(\log N)$개의 정점만 확인한다.

```
#include <bits/stdc++.h>
```

```cpp
using namespace std;
using ll = long long;

int N, Q;
vector<int> Inp[101010], G[101010];

ll T[1 << 18], L[1 << 18];
void Push(int node, int s, int e){
    T[node] += (e - s + 1) * L[node];
    if(s != e) L[node<<1] += L[node], L[node<<1|1] += L[node];
    L[node] = 0;
}
void Update(int l, int r, int v, int node=1, int s=1, int e=N){
    Push(node, s, e);
    if(r < s || e < l) return;
    if(l <= s && e <= r){ L[node] += v; Push(node, s, e); return; }
    int m = s + e >> 1;
    Update(l, r, v, node<<1, s, m);
    Update(l, r, v, node<<1|1, m+1, e);
    T[node] = T[node<<1] + T[node<<1|1];
}
ll Query(int l, int r, int node=1, int s=1, int e=N){
    Push(node, s, e);
    if(r < s || e < l) return 0;
    if(l <= s && e <= r) return T[node];
    int m = s + e >> 1;
    return Query(l, r, node<<1, s, m) + Query(l, r, node<<1|1, m+1, e);
}
int Kth(ll k, int node=1, int s=1, int e=N){
    Push(node, s, e);
    if(s == e) return s;
    int m = s + e >> 1;
    Push(node<<1, s, m); Push(node<<1|1, m+1, e);
    if(k <= T[node<<1]) return Kth(k, node<<1, s, m);
    else return Kth(k-T[node<<1], node<<1|1, m+1, e);
}

int D[101010], P[22][101010], S[101010], Top[101010];
int In[101010], Out[101010], Rev[101010], pv;

void DFS(int v, int b=-1){
    for(auto i : Inp[v]){
        if(i == b) continue;
        D[i] = D[v] + 1; P[0][i] = v;
        G[v].push_back(i);
        DFS(i, v);
    }
}
void DFS1(int v){
    S[v] = 1;
    for(auto &i : G[v]){
        DFS1(i); S[v] += S[i];
        if(S[i] > S[G[v][0]]) swap(G[v][0], i);
    }
}
void DFS2(int v){
    In[v] = ++pv; Rev[pv] =v;
    for(auto i : G[v]){
```

```
        Top[i] = i == G[v][0] ? Top[v] : i;
        DFS2(i);
    }
    Out[v] = pv;
}

ll Total;
void UpdateSub(int v){
    Update(In[v], Out[v], 1);
    Total += Out[v] - In[v] + 1;
}
void UpdatePath(int u, int v){
    for(; Top[u] != Top[v]; u=P[0][Top[u]]){
        if(D[Top[u]] < D[Top[v]]) swap(u, v);
        Update(In[Top[u]], In[u], 1);
        Total += In[u] - In[Top[u]] + 1;
    }
    if(D[u] > D[v]) swap(u, v);
    Update(In[u], In[v], 1);
    Total += In[v] - In[u] + 1;
}
int Query(){
    int v = Rev[Kth((Total + 1) / 2)];
    if(Query(In[v], Out[v]) * 2 > Total) return v;
    for(int i=21; ~i; i--){
        if(!P[i][v]) continue;
        if(Query(In[P[i][v]], Out[P[i][v]]) * 2 <= Total) v = P[i][v];
    }
    return P[0][v];
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<N; i++){
        int s, e; cin >> s >> e;
        Inp[s].push_back(e); Inp[e].push_back(s);
    }
    DFS(1); DFS1(1); DFS2(Top[1]=1);
    for(int i=1; i<22; i++) for(int j=1; j<=N; j++) P[i][j] = P[i-1][P[i-1][j]];

    cin >> Q;
    while(Q--){
        int op, a, b; cin >> op >> a;
        if(op == 1) UpdateSub(a);
        else cin >> b, UpdatePath(a, b);
        cout << Query() << "\n";
    }
}
```